



Banco de dados local com Ionic e imagens

Trabalhando com imagens em Ionic

O framework Ionic permite a criação de aplicativos multiplataforma, que podem ser desenvolvidos para Web, Android e iOS. Isso se torna possível graças à utilização do Capacitor e do Cordova, que possibilitam o acesso às funcionalidades do telefone ou do sistema operacional, como, por exemplo, o sistema de arquivos do dispositivo. No exemplo a seguir, vamos demonstrar como capturar uma imagem e salvá-la na memória do telefone.

Para essa aula vamos usar como base a documentação oficial do Ionic que se encontra nas referências desse material.

Praticando com Imagens e Capturando uma Imagem

Para dar início vamos criar um arquivo de *service* onde vamos criar as funções para trabalharmos com as imagens vamos continuar trabalhando com projeto criado nas aulas anteriores.

Execute o comando abaixo para criar um arquivo de service

ionic g service services/photo

Agora nesse arquivo vamos adicionar as dependências que vamos utilizar nesse pro

```
import { Camera, CameraResultType, CameraSource, Photo } from
'@capacitor/camera';

import { Filesystem, Directory } from '@capacitor/filesystem';

import { Preferences } from '@capacitor/preferences';
```

Para essas dependências funcionarem é necessário instalar as mesmas através do npm.

npm i @capacitor/core

npm i @capacitor/câmera

npm i @capacitor/filesystem

npm i @capacitor/preferences

Agora vamos continuar editando nosso arquivo de service

```
public async addNewToGallery() {

  // Take a photo

  const capturedPhoto = await Camera.getPhoto({

   resultType: CameraResultType.Uri,

   source: CameraSource.Camera,

   quality: 100

  });

 }
```

Essa função vai capturar a imagem da câmera ou dos arqu do computador caso esteja executando no browser.

Agora vamos fazer uma alteração no componente, nesse caso escolhemos o componete tab3.

```
constructor(public photoService: PhotoService) {}

addPhotoToGallery() {

  this.photoService.addNewToGallery();

}
```

Agora vamos editar o HTML desse componente adicionando um fab button.

```
<ion-fab vertical="bottom" horizontal="center" slot="fixed">

  <ion-fab-button (click)="addPhotoToGallery()">

    <ion-icon name="camera"></ion-icon>

  </ion-fab-button>

</ion-fab>
```

Agora temos que criar uma classe para armazenar as fotos temporariamente.

```
export class UserPhoto {

 filepath: string = '';

 webviewPath: string = '';
```

```
}
```



Vamos adicionar uma lista desse objeto no arquivo de service.

```
public photos: UserPhoto[] = [];
```

Agora vamos adicionar as fotos a essa lista.

```
this.photos.unshift({
  filepath: "soon...",
  webviewPath: capturedPhoto.webPath!
});
```

Adicionamos o código acima a função addNewToGallery.

Agora vamos listar todas as fotos em nosso HTML do componente.

```
<ion-grid>
  <ion-row>
    <ion-col size="6" *ngFor="let photo of photoService.photos; index as position">
      <ion-img [src]="photo.webviewPath"></ion-img>
    </ion-col>
  </ion-row>
</ion-grid>
```

Agora já podemos testar a aplicação no Android Studio através dos comandos abaixo:



ionic build

npx cap add android

npx cap open android

Pronto agora temos nossa aplicação rodando, porém ao fechar a mesma vou perder as minhas fotos então para resolver esse problema vamos a seguir salvar as mesmas na memória do telefone.

Salvando a Imagem na Memória do Telefone

Para essa funcionalidade vamos utilizar a biblioteca do capacitor de filesystem que vai permitir ter acesso a memória interna do telefone.

Agora vamos fazer algumas alterações no arquivo de service primeiro vamos alterar a função addNewToGallery novamente.

```
public async addNewToGallery() {

  // Take a photo

  const capturedPhoto = await Camera.getPhoto({

    resultType: CameraResultType.Uri, // file-based data; provides
best performance

    source: CameraSource.Camera, // automatically take a new photo
with the camera
```

```
    quality: 100 // highest quality (0 to 100)
  });

  // Save the picture and add it to photo collection
  const savedImageFile = await this.savePicture(capturedPhoto);
  this.photos.unshift(savedImageFile);
}
```



Agora vamos criar uma função para salvar a imagem e converter a mesma para base64.

```
private async savePicture(photo: Photo) {
  // Convert photo to base64 format, required by Filesystem API to
save
  const base64Data = await this.readAsBase64(photo);


  // Write the file to the data directory
  const fileName = new Date().getTime() + '.jpeg';
  const savedFile = await Filesystem.writeFile({
    path: fileName,
    data: base64Data,
```



```
    directory: Directory.Data
  });


  // Use webPath to display the new image instead of base64 since it's
  // already loaded into memory
  return {
    filepath: fileName,
    webviewPath: photo.webPath!
  };
}
```

Para que essas funções funcionem multiplataforma principalmente na web, vamos ter que criar mais duas funções de conversão para base64.

```
private async readAsBase64(photo: Photo) {
  // Fetch the photo, read as a blob, then convert to base64 format
  const response = await fetch(photo.webPath!);
  const blob = await response.blob();


  return await this.convertBlobToBase64(blob) as string;
```

```
 }
```



```
 private convertBlobToBase64 = (blob: Blob) => new
Promise((resolve, reject) => {
  const reader = new FileReader();
  reader.onerror = reject;
  reader.onload = () => {
    resolve(reader.result);
  };
  reader.readAsDataURL(blob);
 });
```

Carregando as Fotos da Memória do Telefone

Agora vamos carregar as fotos da memória do telefone, para isso vamos alterar novamente nosso arquivo de service.

```
export class PhotoService {
 public photos: UserPhoto[] = [];
 private PHOTO_STORAGE: string = 'photos';
```

Agora no final da função addNewToGallery vamos adicionar o seguinte código.



```
  Preferences.set({
   key: this.PHOTO_STORAGE,
   value: JSON.stringify(this.photos),
  });
```

Agora vamos criar uma função para carregar essas fotos e adicionar a nossa lista de fotos.

```
public async loadSaved() {
  // Retrieve cached photo array data
  const photoList = await Preferences.get({ key:
this.PHOTO_STORAGE });
  this.photos = JSON.parse(photoList.value!) || [];
  // Easiest way to detect when running on the web:
  // "when the platform is NOT hybrid, do this"
  if (!this.platform.is('hybrid')) {
   // Display the photo by reading into base64 format
   for (let photo of this.photos) {
    // Read each saved photo's data from the Filesystem
    const readFile = await Filesystem.readFile({
     path: photo.filepath,
```

```
      directory: Directory.Data

    });

    // Web platform only: Load the photo as base64 data

    photo.webviewPath = data:image/jpeg;base64,</span><span
style="font-size:20px;color:#569CD6">${</span><span
style="font-size:20px;color:#4FC1FF">readFile</span><span
style="font-size:20px;color:#D4D4D4">.</span><span
style="font-size:20px;color:#9CDCFE">data</span><span
style="font-size: 20px;color:#569CD6">}</span><span
style="font-size:20px;color:#CE9178">;

   }

  }

 }
```



Agora no nosso componente vamos executar essa função.

```
export class Tab3Page {

 constructor(public photoService: PhotoService) {}

 async ngOnInit() {

  await this.photoService.loadSaved();

 }

 addPhotoToGallery() {

  this.photoService.addNewToGallery();
```



Agora vamos fazer algumas pequenas alterações no código para que fique 100% compatível com as aplicações móveis.

Novamente no service vamos adicionar uma variável para definir a plataforma.

```
constructor(private platform: Platform
```

Agora vamos alterar a função readAsBase64

```
private async readAsBase64(photo: Photo

  // "hybrid" will detect Cordova or Capacitor

  if (this.platform.is('hybrid')

    // Read the file into base64 format

    const file = await Filesystem

      path: photo.path

    return file.data

  else

    // Fetch the photo, read as a blob, then convert to base64 format
```



```
    const response = await fetch(photo.webPath!);
    const blob = await response.blob();
    return await this.convertBlobToBase64(blob) as string;
  }
}
```

E também nossa função savePicture o que basicamente estamos fazendo é detectando se a aplicação está rodando para executar determinada ação caso seja um telefone Android ou iOS.

```
 private async savePicture(photo: Photo) {
   // Convert photo to base64 format, required by Filesystem API to
save
   const base64Data = await this.readAsBase64(photo);
   // Write the file to the data directory
   const fileName = new Date().getTime() + '.jpeg';
   const savedFile = await Filesystem.writeFile({
     path: fileName,
     data: base64Data,
     directory: Directory.Data
   });
```

```
  // Use webPath to display the new image instead of base64 since
it's

  // already loaded into memory

  if (this.platform.is('hybrid'

    // Display the new image by rewriting the 'file://' path to HTTP

    // Details:
https://ionicframework.com/docs/building/webview#file-protocol

    return

      filepath: savedFile.uri

      webviewPath: Capacitor.convertFileSrc(savedFile.uri

  else

    // Use webPath to display the new image instead of base64 since
it's

    // already loaded into memory

    return

      filepath: fileName

      webviewPath: photo.webPath
```



Pronto, agora já temos a aplicação de galeria de fotos já funcionando.

Agora para rodar novamente executamos os seguintes comandos.

ionic build

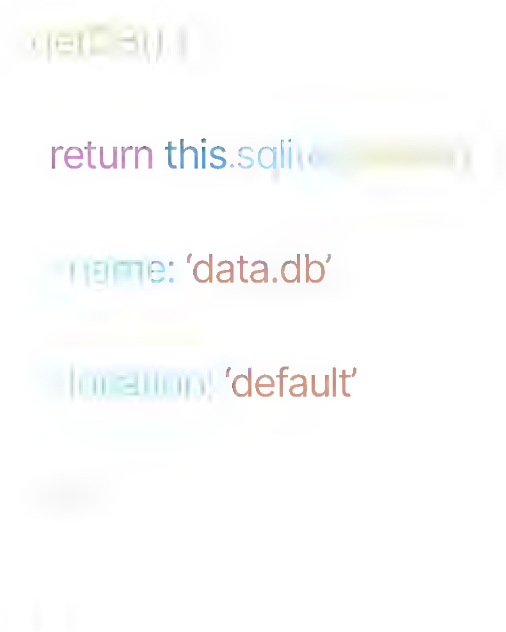
npx cap sync android

npx cap open android

Declarando SQLite e adicionando a imagem base64 no SQLite

Agora para complementar nosso projeto vamos salvar essa lista de fotos em uma tabela do SQLite, nossa foto já está sendo armazenada, porém não está em um banco SQLite.

Vamos continuar usando o modelo das aulas anteriores.



Vamos criar uma tabela de fotos.



```
createTable() {

  const SQL = 'CREATE TABLE photos( id INTEGER PRIMARY KEY, fileName VARCHAR(120), path VARCHAR(120));'

  this.getDB().then((db: SQLiteObject) => {

   db.executeSql(SQL, [])

     .then(() => console.log('Executed SQL'))

     .catch(e => console.log(e));

  })

  .catch(e => console.log(e))

}
```

Agora vamos inserir todas as fotos da galeria.

```
saveToDB() {

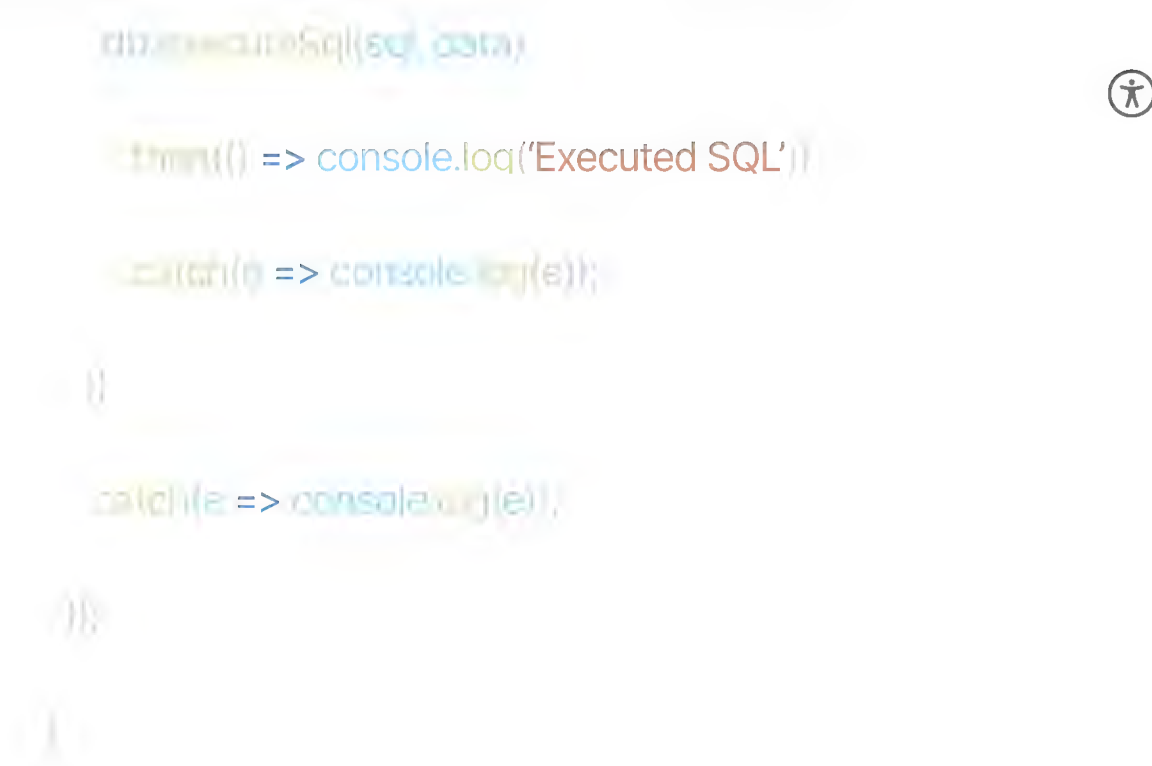
  let count = 0

  this.photoService.getPhotos().forEach(photo => {

   let [illegible] = 'insert into photos (fileName, path) values (?,?)'

   let data = ['photo'+count, photo.filepath]

   this.getDB().then((db: SQLiteObject) =>
```



Posteriormente vamos alterar esse código para o contexto de postos de gasolina onde vamos criar um avatar da foto dos usuários do app.

Atividade Extra

Para se aprofundar no assunto desta aula assista o vídeo de SQLite (O Banco de Dados de Bolso) // Dicionário do Programador do Código Fonte TV.

Canal: Código Fonte TV

Título a ser buscado no youtube: **SQLite (O Banco de Dados de Bolso) // Dicionário do Programador**



Referência Bibliográfica

IMPERIAL, Juliana. **Banco de Dados para Dispositivos Móveis** . SQLite - **The Architecture of SQLite**. <http://sqlite.org/arch.html>. Acesso em: 20 jun. 2022.

IONIC Framework. **IONIC**. Disponível em: <https://ionicframework.com>. Acesso em: 30 nov. 2022.

Atividade Prática 13 - Adicionando Suporte para Banco de Dados

Título da Prática: Criação de uma implementação do Banco de Dados SQLite

Objetivos: Utilizar o banco de dados SQLite para testar e implementar as execuções da atividade prática.

Materiais, Métodos e Ferramentas: Iremos fazer uma pesquisa na internet e utilizar o banco de dados SQLite.

Atividade Prática

O uso do SQLite é recomendado onde a simplicidade da administração, implementação e manutenção são mais importantes que vários recursos que SGBDs mais voltados para aplicações complexas implementam. E essa simplicidade é amplamente requisitada hoje em dia. (BICALHO 2014).

Na prática, o SQLite é capaz de criar um arquivo em disco, ler e escrever diretamente sobre este arquivo. O arquivo criado possui a extensão ".db" e é capaz de manter diversas tabelas. Uma tabela é criada com o uso do comando CREATE TABLE da linguagem SQL. Os dados das tabelas são manipulados através de comandos DML (INSERT, UPDATE e DELETE) e são consultados com o uso do comando SELECT (GONÇALVES, 2011).

Sabemos que o núcleo da infraestrutura SQLite contém o usuário interface, o processador de comandos SQL e a máquina abstrata (SQLite,2019).

Já a partir dessas informações, a nossa proposta agora é gerar uma saída dentro do banco de dados SQLite para fazer uma tabela que irá armazenar arquivos de imagem para serem manipuladas por qualquer sistema de preferência com Ionic após gravar no SQLite.

Gabarito Atividade Prática

Primeiro vamos criar nossa tabela para salvar as fotos.

```
CREATE TABLE photos
( id INTEGER PRIMARY KEY,
  fileName VARCHAR(120),
  path VARCHAR(255));
```

Agora vamos para código para criar essa tabela no IONIC



```
createTable(){
  const SQL = 'CREATE TABLE photos( id INTEGER PRIMARY KEY, fileName VARCHAR(120), path VARCHAR(120));'
  this.getDB().then((db: SQLiteObject) => {
   db.executeSql(SQL, [])
    .then(() => console.log('Executed SQL'))
    .catch(e => console.log(e));
  })
  .catch(e => console.log(e));
 }
```

Você pode consultar o material da apostila para verificar como fazer a captura da imagem. Agora vamos salvar os dados nessa tabela através de uma função.

```
 saveToDB(){
  let count = 0;
  this.photoService.getPhotos().forEach(photo => {
   let sql = 'insert into photos (fileName, path) values (?,?)';
   let data = ['photo'+count, photo.filepath];
   this.getDB().then((db: SQLiteObject) => {
```



```
        db.executeSql(sql, data)
          .then(() => console.log('Executed SQL'))
          .catch(e => console.log(e));
      })
      .catch(e => console.log(e));
    });
  }
```

Ir para exercício